(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 817 032 A2

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
07.01.1998 Bulletin 1998/02

(51) Int Cl.6: **G06F 9/46**, G06F 9/44

(21) Application number: **97304092.6**

(22) Date of filing: **12.06.1997**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: **03.07.1996 US 675563**

(71) Applicant: **SUN MICROSYSTEMS, INC.**
Mountain View, California 94043-1100 (US)

(72) Inventors:
- **Fowlow, Brad G.**
Redwood City, California 94062 (US)
- **Nuyens, Greg B.**
Menlo Park, California (US)
- **Messer, Keith L.**
San Francisco, California 94112 (US)
- **Ludolph, Frank**
Menlo Park, California 94025 (US)

(74) Representative: **Browne, Robin Forsythe, Dr.**
Urquhart-Dykes & Lord
Tower House
Merrion Way
Leeds LS2 8PA West Yorkshire (GB)

(54) **Cataloging apparatus for facilitating the re-use of distributed objects in a distributed object system**

(57) A method, apparatus, and computer program product for selecting and reviewing a distributed object installed on a distributed object system is described. In one embodiment, the method of the invention includes generating a library of components corresponding to distributed objects on said distributed object system, including one component corresponding to the distributed object. Each of the components of the library includes information describing the distributed object to which the particular component corresponds. The contents of the library are displayed using a catalog interface device. The library is browsed using the catalog interface device to identify the component corresponding to the distributed object, and is selected. At least a portion of the information describing the distributed object is displayed.

EP 0 817 032 A2

Description

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The present invention relates to the fields of distributed computing systems, client-server computing and object-oriented programming. More specifically, the present invention includes a method and apparatus for creating object-oriented software applications for use on a distributed object system.

2. The Relevant Art

Object-oriented programming methodologies have received increasing attention over the past several years in response to the increasing tendency for software developed using traditional programming methods to be delivered late and over budget. Object-oriented programming methodologies focus on manipulating data rather than procedures; thus providing the programmer with a more intuitive approach to modeling real world problems. In addition objects encapsulate related data and procedures so as to hide that information from the remainder of the program by allowing access to the data and procedures only through the object's interface. Hence changes to the data and/or procedures of the object are relatively isolated from the remainder of the program. This provides code that is more easily maintained as compared to code written using traditional methods, since changes to an object's code do not affect the code in the other objects. In addition. the inherent modular nature of objects allows individual objects to be re-used in different programs. Thus, programmers can develop libraries of "tried and true" objects that can be used over and over again in different applications. This increases software reliability while decreasing development time, as reliable programming code may be used repeatedly.

Object-oriented distributed systems are based upon a client-server model, in which object-servers provide interfaces to clients that make requests of the object-servers. Typically, these servers are objects consisting of data and associated methods. The clients obtain access to the functionalities of the object-servers by executing calls on them, which calls are mediated by the distributed system. When the object-server receives the call it executes the appropriate method and transmits the result back to the object-client. The client and object-server communicate through an Object Request Broker (ORB) which is used to locate the various distributed objects and establish communications therebetween.

Although the advantages to employing object-oriented programming methodologies through distributed object systems are significant, there remain major hurdles to their implementation. In general, the goal of implementing the re-use of software during the programming process. even in the context of object programming, is difficult to achieve. Typically, programmers are reluctant to use code about which their understanding is minimal or even nonexistent. This is compounded in distributed object systems as the developer(s) of the code may not be easily reached to provide comments and instruction to the programmer whose task is to develop a new application. Thus, although much useful code may be available to the programmer throughout the distributed object system that programmer may not be able to take full advantage of it, thus being forced to rewrite sections of code that have already been developed.

Present methodologies for sharing code are inadequate in the distributed object model. Current methods for sharing object code include the use of shared libraries. publicly available directories of header files, and the wide-spread distribution of documentation (either electronically or by hard copy). These methods, however, do not lend themselves well to a distributed object environment in which the basic elements being manipulated are not file-based but rather are interface-based. Also, methods for re-using objects in a distributed object system should seek to increase the user's sense of a "community" of programmers and the code they produce. Thus, systems seeking to increase the re-use of objects in a distributed object system should facilitate the user's determination of an object's identity, its purpose, and its method of use.

Another challenge to programming in a distributed object environment is the need to provide large amounts of "boilerplate" type computer code so that objects and applications developed for use in the distributed object system can function properly within that system; in particular, the need to provide basic computer code structures to enable ordinary objects (*i.e., C++* objects) to function as distributed objects. In addition, however, the programmer seeking to maximize the re-use of existing objects and computer code in the distributed objects system is faced with similar although slightly different challenges. When a programmer wishes to employ a object in an application being developed, the programmer must indicate basic information pertaining to the object such as calls to the object's factory and provide various initialization data. In addition. a variety of make files, header files, and library dependencies must be accounted for in the code before that code is forwarded to additional facilities. Additional details that must be provided by the programmer include various exception handling, debugging, and tracing support code. Again, although the methods and code required are generally known to those of skill in the art, the implementation of such routines is laborious, repetitive, and prone to error. Thus, the development of appropriately coded object applications is an extremely time consuming and painstaking task. It would therefore be preferable to automate the incorporation of such "housekeeping" code.

Thus, it would be desirable to allow programmers

and other users the ability to create and install distributed objects in a relatively transparent fashion so that objects created in different programming languages and/or objects residing on different computing platforms can be made available on distributed object systems without extensive re-invention of existing programming code, or placing an undue burden on the user. Solutions to these problems would facilitate the development of object-oriented applications and distributed object applications, by allowing the programmer to focus on the tasks that required real creative effort and minimize repetitive coding and analysis.

SUMMARY OF THE INVENTION

The present invention provides methods, apparatus, and computer program products described herein allow programmers to access and review distributed objects already installed on a distributed object system to determine whether such pre-existing objects can be included in software that is under development. As will be described in greater detail herein, the methods, apparatus, and computer program products for facilitating the re-use of distributed objects installed on a distributed object system, enhance the efficiency of the software construction process and the reliability of the code so produced.

In one aspect, the present invention provides a computer-implemented method for selecting and reviewing a distributed object installed on a distributed object system. In one embodiment, the method of the invention includes generating a library of components corresponding to distributed objects on said distributed object system, including one component corresponding to the distributed object. Each of the components of the library includes information describing the distributed object to which the particular component corresponds. The contents of the library are displayed using a catalog interface device. The library is browsed using the catalog interface device to identify the component corresponding to the distributed object, and is selected. At least a portion of the information describing the distributed object is displayed.

In one embodiment of the above-described method, displaying the contents of said library comprises providing a display of at least a portion of the hierarchical directory structure of the library. In another embodiment, the step of browsing said library of components comprises selecting a file path using said display of said hierarchical directory structure of said library and displaying representations of components located in said path. In still another embodiment. the representations comprise icons, and said step of selecting comprises making a selection action on an icon.

In another aspect, the present invention provides a computer system for selecting and reviewing a distributed object installed on a distributed object system. In one embodiment of the computer system provided by the invention, the system comprises a library of components corresponding to distributed objects on said distributed object system, wherein each of said components includes information describing the distributed object to which the component corresponds. The library of components includes one component corresponding to the distributed object. The computer system of the invention further includes a library manager which is effective to locate and retrieve said components. Finally, a catalog interface device for displaying, reviewing and selecting said components in said library is provided. The catalog interface device is coupled with said library manager such that queries and selection actions made in said catalog interface device are communicated to said library manager.

In one embodiment, the computer system of the invention further includes a component generator which is effective to generate components corresponding to selected. installed distributed objects in said distributed objects system. In a more specific embodiment, the component generator is coupled with the distributed objects to which said components refer. In another embodiment, the computer system of the invention includes a builder coupled with said component generator such that said components are generated when the distributed objects to which the components refer are processed by said builder. In still another embodiment, the catalog interface comprises a graphical user interface. In a more particular embodiment, the catalog interface devices comprises a display of at least a portion of the hierarchical directory structure of said library.

In still another embodiment, the computer system provided by the present invention comprises a processing unit which is coupled to an input/output device. Random access memory is also coupled with the processing unit. A memory device is in communication with the processing unit. The memory device includes a distributed object reference data structure and an associated component data structure, said component data structure including presentation information, programmer information, and tool information relating to said distributed object data structure. In one embodiment, the computer system further includes a library of components contained in a library data structure residing in said memory device, said library of components including a library service effective to locate and retrieve said components in response to commands received from said input/output device.

In yet another aspect, the present invention provides a computer program product comprising a computer-usable medium having computer-readable program code embodied thereon for a component data structure, said component data structure being related to a distributed object data structure installed on a distributed object system, and said component data structure including presentation information, programmer information, and tool information relating to said distributed object data structure.

In one embodiment, the computer readable program code devices include code devices effective for generating the component in addition to generating a library of component data structures corresponding to distributed object data structures on said distribute object system, wherein each of said component data structures includes information describing a distributed object data structure to which the component data structure corresponds. This embodiment also includes computer readable program code devices configured to cause a computer to display the contents of the library using the catalog interface device to identify said distributed object data structures, browsing said library of component data structures using said catalog interface device to identify said distributed object data structures, and displaying at least a portion of said information describing said distributed object data structures.

These and other aspects and advantages of the present invention will become more apparent when the Description below is read in conjunction with the accompanying Drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a schematic illustration of an object request broker (ORB) in accordance with the present invention.

Figure 2 is an illustration of a computer network in accordance with the present invention.

Figure 3 is a schematic illustration of a computer system in accordance with the present invention.

Figure 4 is a schematic illustration of a system for constructing object-oriented applications in a distributed object system in accordance with the present invention.

Figure 5 is a schematic illustration of a hierarchical directory structure for storing, searching, and retrieving components corresponding to distributed objects in accordance with the present invention.

Figure 6 is an illustration of a graphical user interface for searching and examining components according to one embodiment of the present invention.

DESCRIPTION OF SPECIFIC EMBODIMENTS

Section 1 below describes the basic organization and operation of distributed object systems. Second 2, at page 11 below, describes the components, library services, and uses thereof, as provided by the present invention.

1. PHYSICAL EMBODIMENTS AND BACKGROUND OF DISTRIBUTED OBJECT SYSTEMS

The present invention employs various process steps involving data stored in computer systems. These steps are those requiring physical manipulation of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It is sometimes convenient, principally for reasons of common usage, to refer to these signals as bits, values, elements, variables, characters, data structures, or the like. It should be remembered, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

Further, the manipulations performed are often referred to in terms such as identifying, running. or comparing. In any of the operations described herein that form part of the present invention these operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or other similar devices. In all cases, there should be borne in mind the distinction between the method of operations in operating a computer and the method of computation itself. The present invention relates to method steps for operating a computer in processing electrical or other physical signals to generate other desired physical signals.

The present invention also relates to an apparatus for performing these operations. This apparatus may be specially constructed for the required purposes, or it may be a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. The processes presented herein are not inherently related to any particular computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may be more convenient to construct a more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description given below.

In addition, the present invention further relates to computer readable media which include program instructions for performing various computer-implemented operations. The media and program instructions may be those specially designed and constructed for the purposes of the present invention, or they may be of the kind well known and available to those having skill in the computer software arts. Examples of computer readable media include, but are not limited to. magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and hardware devices that are specially configured to store and perform program instructions, such as read-only memory devices (ROM) and random access memory (RAM). Examples of program instructions include both machine code, such as produced by a compiler, and files containing higher level code that can be executed by the computer using an interpreter.

A distributed object system 10 typically includes an Object Request Broker (ORB) 11 as is symbolically illustrated in Figure 1. ORB 11 provides location and

transport mechanisms and facilities necessary to deliver a call from a client to a servant (target object) and to return a response to the client. The client and servant may be located in the same process, in different processes on the same machine, or on completely different machines. For the purposes of this discussion, client 20 may be any code that invokes an operation on a distributed object and thus may or may not take the form of a distributed object or a process. A distributed object may have a wide variety of representations. By way of example, the distributed object may be a C++ object that has been provided by an application developer. Alternatively, an implementation for a distributed object may be developed within a visual application builder 15 as described in greater detail in Section 2 below. In brief, the visual application builder allows a developer to visually select existing object types from a catalog of object types available on the distributed object system and graphically connect the services provided by the selected objects to create a new implementation for an object.

An object development facility 16 may be used to simplify the creation and the installation of distributed objects, in part, by "wrapping" or encapsulating developer objects in distributed object code. As such, object development facility 16 may be used to transform a developer object into an ORB object implementation 14. In this example, ORB object implementation 14 is presented as a server as shown by its location in the diagram. A developer uses an interface definition language to define an interface for an ORB object, provides a developer object implementation that implements that object's behavior, and then uses the object development facility 16 in order to produce an ORB object implementation 14. At run time, an instance of this ORB object (a servant object) is created that will utilize this ORB object implementation 14. It should be appreciated that the object development facility may also be used to create objects that take the role of clients at some point.

Client 20 communicates with a servant by way of a stub 21, a subcontract layer 36, possibly a filter 40, and a transport layer 38. Stub 21 includes a surrogate 22, a method table 24 and stub functions 25. Client 20 communicates initially with surrogate 22 that appears to the client as the servant object. Alternatively, client 20 may communicate directly with the servant object through a dynamic invocation interface (DII) 26 instead of through surrogate 22, method table 24 and stub functions 25. Dynamic invocation interface 26 is used to enable clients to construct dynamic requests.

Subcontract layer 36 provides the functionality required by an object in order to utilize subcontracts to implement various services (or features or object mechanisms) named by a particular subcontract. A subcontract identifies a quality of service provided by the distributed object system that may be utilized by an individual object. For example, a subcontract may identify that the feature of security is to be used for a particular object. Filter 40, if being used, may perform a variety of tasks, such as compression, encryption, tracing, or debugging, that are to be applied to communications to and from an object. Transport layer 38 operates to marshal, unmarshal and physically transport information to and from a servant that typically does not share the same process as a client.

A standard implementation suite 28 (or object adapter) represents a set of subcontracts that interact with ORB objects 14 in identical ways, as for example object key management. It should be noted that a subcontract may belong to multiple implementation suites. Also, implementation suites may utilize different subcontracts. A skeleton, that may take the form of either static skeleton 32 or dynamic skeleton 30, is used to transform requests into a format required by a servant object 78. Thus, skeletons 30 and 32 call an appropriate servant object 78. Static skeleton 32 is used to call interface-specific object implementations 14, while dynamic skeleton 30 is used generically when interface-specific objects are not available. An ORB interface 34 is the interface that goes directly to the ORB that is the same for all ORBs and does not depend upon an object's interface or object adapter. An ORB daemon 46 is responsible for ensuring that object servers are active when invoked by clients.

Secure Protocol 42 is a secure interoperability protocol that secures the internet inter-ORB protocol and helps to transmit information through transport layer 38 in a secure fashion. This may mean integrity protection, confidentiality, etc. The internet inter-ORB protocol is a protocol that typically communicates between processes on different machines. However, in some cases, the internet inter-ORB protocol may communicate between processes on the same machine. The security server 54 is a security administration server that secures the services that are used between processes on different computers.

Typecode/Any module 44 implements "Typecode" and "Any" objects. Typecode describes an Interface Definition Language (IDL) data type, allowing type descriptions to be transmitted between clients and servers. An instance of an IDL data type may be encapsulated by an Any object. An Any object refers to typecode of the encapsulated data, and a generic encoding of the data.

An implementation repository 50 is used to store information relating to object servers. Specifically, implementation repository 50 stores the information needed to start a server process. For example, implementation repository 50 stores information such as the location of the server program, any arguments to the program, and any environment variables to pass to the program, etc.

Simple persistence 56 uses an Interface Definition Language (IDL)-defined type and the output from running that IDL type through the IDL compiler, together with a portion of additional code so that an IDL-defined type can be read from, and written to, disk. A naming service 52 is used to name ORB objects. A client may

use naming service 52 to find a desired object by name. Naming service 52 returns an object reference, that in turn may be used to send request to that object. An Interface Repository 48 (IFR) knows about all interfaces for all objects within the distributed object system.

In an embodiment of the present invention, distributed objects are located on one or more computers linked together by a computer network such as the network illustrated at 100 in Figure 2. As seen in the Figure, network 100 includes computer 102 which computer is coupled to a network 104. Network 104 can further include a server, router or the like 106 in addition to other computers 108, 110, and 112 such that data and instructions can be passed among the networked computers. The design, construction and implementation of computer networks will be familiar to those of skill in the art.

Computers 102, 106, 108, 110, and 112 are illustrated schematically with respect to Figure 3 at 200. Each computer includes a processing unit 202 effective for performing computations, such as, but not limited to, a central processing unit (CPU), or multiple processors including parallel processors or distributed processors. Processor 202 is coupled with primary memory 204 such as random access memory (RAM) and read only memory. Typically, RAM includes programming instructions and data, including distributed objects and their associated data and instructions, for processes currently operating on processor 202. ROM typically includes basic operating instructions, data and objects used by the computer to perform its functions. In addition, a secondary storage device 208, such as a hard disk, CD ROM, magneto-optical (floptical) drive, tape drive or the like, is coupled bidirectionally with processor 202. Secondary storage device 208 generally includes additional programming instructions, data and objects that typically are not in active use by the processor, although the address space may be accessed by the processor, *e.g.*, for virtual memory or the like. Each of the above described computers further includes an input/output source 210 that typically includes input media such as a keyboard, pointer devices *(e.g.,* a mouse or stylus) and the like. Each computer can also include a network connection 212. Additional mass storage devices (not shown) may also be connected to CPU 202 through network connection 212. It will be appreciated by those skilled in the art that the above described hardware and software elements, as well as networking devices, are of standard design and construction.

The computer-implemented methods described herein can be implemented using techniques and apparatus well-known in the computer science arts for executing computer program instructions on computer systems. As used herein, the term "computer system" is defined to include a processing device (such as a central processing unit, CPU) for processing data and instructions that is coupled with one or more data storage devices for exchanging data and instructions with the processing unit, including, but not limited to, RAM, ROM, CD-ROM, hard disks. and the like. The data storage devices can be dedicated, *i.e.,* coupled directly with the processing unit, or remote. *i.e.*, coupled with the processing unit. over a computer network. It will be appreciated that remote data storage devices coupled to a processing unit over a computer network can be capable of sending program instructions to a processing unit for execution on a particular workstation. In addition, the processing device can be coupled with one or more additional processing devices. either through the same physical structure *(e.g.,* in a parallel processor), or over a computer network (*e.g.*, a distributed processor.). The use of such remotely coupled data storage devices and processors will be familiar to those of skill in the computer science arts. The term "computer network" as used herein is defined to include a set of communications channels interconnecting a set of computer systems that can communicate with each other. The communications channels can include transmission media such as, but not limited to, twisted pair wires, coaxial cable, optical fibers, satellite links, or digital microwave radio. The computer systems can be distributed over large, or "wide" areas *(e.g.,* over tens, hundreds, or thousands of miles. WAN), or local area networks *(e.g.,* over several feet to hundreds of feet, LAN). Furthermore, various local- and wide-area networks can be combined to form aggregate networks of computer systems. One example of such a confederation of computer networks is the "Internet".

Figure 4 at 400 illustrates schematically one embodiment of a system for composing and installing object-oriented applications in a distributed object system. The illustrated system includes a composition builder 402 which the user, typically a programmer, employs to compose applications for installation on the distributed object system such as that shown in Figure 1 described above. The composition builder is coupled with a component service 404, described in greater detail hereinbelow, that provides the user or programmer access to objects available on the distributed object system. Composition builder 402 is further coupled to a code generator 408 which, in conjunction with program template repository 406, takes the composition created by the composition builder and produces program source files as shown at 410.

Programs source files 410 are then forwarded to ODF compiler/linker 414. In addition. ODF compiler/linker 414 is coupled with object access software 412 which object access software is coupled with component service 404. Object access software 412 comprises a set of stubs that are produced when an IDL compiler is used to prepare an language mapping for an IDL interface in accordance with OMG CORBA specifications for the construction of distributed objects. ODF compiler/linker 414 produces both object software 416 and network applications 418 which in turn access network objects 422 as shown generally at 420. These network objects can then employed by the object access software 414 either

for use in the catalogue service 404 or in conjunction with ODF compiler/linker 414 as indicated by the dashed arrows.

2. LIBRARIES OF COMPONENTS REPRESENTING DISTRIBUTED OBJECTS TO FACILITATE RE-USE OF DISTRIBUTED OBJECTS

In one aspect, the present invention provides a development environment and devices adapted to facilitate the re-use of distributed object code in a distributed object system. An illustration of one embodiment of such an environment is provided at 500 in Figure 5, which illustrates a hierarchical directory structure through which users (typically programmers) can browse, examine, and access components that represent distributed objects available for re-use across the distributed object system. Thus, it will be recognized that using the hierarchical directory structure and library such as illustrated in Figure 5 programmers will have access to code located at various points throughout the distributed object system through a central library of references to that code.

Referring now to hierarchical directory structure 500 in greater detail, it will be appreciated by those of skill in the art that the illustrated structure is representative of a naming service commonly used in distributed object systems. At 502 a workgroup root directory is defined, from which root directory 502 extends subdirectories defining contexts such as context 504. A particular context is a Services context shown at 506 from which Services context extends various subdirectories indicating services available on the network including Component service 508. From Component service subdirectory 508 extends various services and components such as Library Service 510, Components subdirectory 512 and Incidental Objects subdirectory 514. Components subdirectory 512 includes a Naming context subdirectory 516 from which extends the individual components that represent distributed objects throughout the distributed object system such as shown at 518, 520, and 522.

Library service 510 is in communication with various Catalog clients, such as shown schematically at 524, through which users of the system access the Library Service to locate and access components such as those at 518, 520 and 522. In one embodiment, Library Service 510 manages Components subdirectory 512, and its subdirectories, as indicated by the dashed box at 524. Thus, in this embodiment the Library Service allows the user access to all of the components within the Components subdirectory 512 thereby providing a central location across which components can be searched and their related distributed objects identified and obtained for re-use in the development of new distributed object applications. In one embodiment, the Library Service is an distributed object acting as a server to provide a single "librarian" for the components, thereby giving consistency and managed access for multiple users across the network. In a particular embodiment, the library service provides both a file system and a query interface through which the names and various aspects of the components, such as their interfaces, can be searched by a user.

In the embodiment illustrated in Figure 5, the present invention utilizes component objects (also referred hereinbelow to as "components") that represent distributed objects in the system. The components are made available through, *e.g.,* Library Service 510 using Catalog Client 524, for users to search and browse to determine the nature of the distributed objects related to the components. It will be appreciated by those of skill in the distributed computing arts that such an arrangement offers advantages over a system in which references are made directly to the distributed objects. In particular, the distributed objects by their very nature are highly transient entities. In some case these objects are being modified and moved within the system so that it is difficult for any sort of library service, such as shown at 510, to maintain a record of the location of each distributed object without incurring large computational overhead.

It will also be appreciated that in a typical browsing action the user does not necessarily require access to the entirety of the distributed object immediately. Typically. the user is seeking more general information about the basic nature of the object and how the code within that object is used so that they may determine whether they desire access to the underlying distributed object in their code. In one embodiment, the present invention satisfies these needs by providing components that represent the build time descriptions of the distributed object to which each component refers. Thus, it will be recognized that the components provided by the invention act as place holders for distributed objects. The actual objects can be accessed at appropriate times during the construction of new applications.

In one embodiment, the components themselves are objects and therefore are defined by an interface, such as an IDL interface, that provides the user with a description of the attributes and operations that define the available services provided by the underlying distributed object. In a particular embodiment, each component provides presentation information about the distributed object. In general, this information can be any information which helps to describe the screen presence of the distributed object being referred to by the component. Such presentation information includes, but is not limited to, the identity of any icon or icons used to identify the object on the distributed object system and any names used to refer to the distributed object. In one embodiment, two names are provided for each distributed object: a short name and a long name. The short name can be used, for example, by the file system to identify the object while the long name can be a more descriptive name to assist in the browsing and understanding of the nature of the object. Other types of pres-

entation information include, but are not limited to, various icons used to reflect the state of the object *(e.g.,* icons reflecting selected and deselected states, error states, and object type) as well as sounds.

In addition, in the described embodiment each component includes programmer information to assist the user or programmer in understanding the character of the code in the underlying distributed object. In one such embodiment, the programmer information comprises a data sheet provided in a suitable format, such as rich text format (RTF) or hypertext markup language (HTML), which is embodied in a data stream that is kept within the component. As will be familiar to those of skill in the distributed computing arts, a data stream is an array of bytes that are stored within an object and forwarded in response to requests for the information contained within the array. It will be appreciated that such an arrangement is advantageous over a file-based reference system as the need for hard addressing of files is eliminated. However. it will also be appreciated that a file-based method of providing data sheets is compatible with the scope of the present invention. Such an alternate embodiment could also be implemented by including pointers to files kept in a database. Examples of the types of programmer information contained in the data sheet include keywords to allow to facilitate searching across the library of components for components representing objects having certain characteristics, and "hard" information that provides the user with data for compiling and using the underlying software.

In one embodiment, the hard information contained in the programmer information data sheet includes, but is not limited to, the interface definition of the underlying object, the factory interface definition for the object, the name or names of the methods used to acquire instances of the underlying object *(e.g.,* the factory method name), the service name of the factory that is installed when the object is created, and the various plugs, sockets and properties that are available in the underlying distributed object. In addition, various installation information is included with the data sheet such as the names of header files for both the object and interface definitions of the distributed object, the locations of the header files, information describing necessary prerequisites to obtain the code required to compile and build the object, as well as macro definitions and compiler information.

Components can be constructed using a variety of methods. In one embodiment, the components are constructed during the building, packaging, and installation process performed when the object is constructed. In this embodiment, a populator mechanism is used to take a text representation of the component data which is extracted from the object during the build and packaging stages to define and install the corresponding component automatically. It will be appreciated by those of skill in the programming arts that such a process greatly facilitates the re-use of code by making the generation of components relatively transparent to the programmer by relieving the programmer of the need to consciously construct a component by hand or otherwise initiate a separate process for the building and installation of components. Alternatively, the component can be constructed in a similar fashion through a builder using a mechanism similar to that just described. In still another embodiment the component interface can be defined through a service offered by the distributed object system so that a programmer can incorporate the service through the inheritance structure of the distributed object such that users can call on an object to have a component built from the object itself. Such an embodiment offers the advantage of providing programmers customizable methods of constructing components. The above-described methods can be implemented using standard techniques known to those of skill in the object programming arts and especially those of skill in the distributed object programming arts.

One embodiment of catalog client 524 is illustrated at 600 in Figure 6. This embodiment is an illustration of a graphical user interface (GUI) that can be incorporated with a visual application builder. Figure 6 illustrates a cataloger 600 which includes a header bar 602, a context browser 604, a separator region 606, a component catalog viewer 608, a second separator region 610, and a detail viewer region 612. Within header region 602 are contained buttons 614, 616, 618 and 620. In one embodiment buttons 614 and 616 control the display in region of 604. For example, depressing button 614 provides the illustrated browser view while depressing button 616 provides a query window into which query window search terms and other search parameters may be entered so that region 524 of Figure 5 can be searched for components having desired properties. Similarly, buttons 618 and 620 in one embodiment are used to control the display of region 612. For example, button 618 when depressed can provide a view of data sheets for components that are selected in region 608 while, if button 620 is depressed, the representation of components that incorporate a component selected in region 608 is displayed.

In the illustrated embodiment, in which region 604 provides a browser over which the context hierarchy such as shown in region of 524 of Figure 5 is shown, the user is presented with various columns 622, 624, 626, and 628 which contain the various contexts and subcontexts of the context structure. In the illustrated example, region 622 contains the contexts "Demo" 630. "Printing" 632 and "Math" 634. These represent the higher levels of the context hierarchy. In the illustration the context "Printing" 632 is shown as being selected, as indicated by the highlighted region surrounding that particular context. Within region 624 are shown those subcontexts that are directly subordinate to the selected context 632. These subcontexts include various printer types such as "Daisywheel" 635, "PostScript" 636, and "Line Printer" 637. A selection of "PostScript" context 636 provides

a second set of subcontexts which depend directly from the PostScript selection as shown in region 626. These subcontexts include the selections "Sun", "Apple", "IBM" and "HP". The PostScript type "Sun" is illustrated as being selected. In the illustrated example, the choices in region 626 represent the lowest subcontext in the particular region of the context hierarchy being shown and, therefore, region 628 is empty. However. were the subcontext selected in region 626 to have subordinate subcontexts those subcontexts would appear in region 628.

Once a selection is made in region 626 (or the region listing the lowest-level contexts of the context hierarchy) a display of all components available within that subcontext are displayed in region 608 (in the case where the appropriate display alternative is selected). These components include "PS Printer" component 640 and shown as highlighted by highlighting box 642 and "Line Printer" component 644 (not highlighted). The selection of "PostScript" component 640, as indicated by border 642, provides a display of the data sheet "PS" 646 in region 612 (again, in the case where the appropriate display alternative is selected). In one embodiment. regions 604, 608, and 612 are scrollable so that information that extends beneath separator regions 606, 610, and the lower end of the window at 612, can be observed by the user. The construction of such a window can be performed using techniques and software familiar to those of skill in the art of graphical user interface programming. Typically operating systems for constructing GUI interfaces include, but are not limited to, *OpenStep* (available from Sun Microsystems, Mountain View, CA), *X Windows* (X Consortium, Cambridge, MA), *Windows* (Microsoft Corporation. Redmond. WA) and the Macintosh operating system (Apple Computer, Cupertino, CA).

The use of components, and the information and links they contain to the underlying objects they represent, greatly facilitates not only the location of existing code for re-use. but also the development of distributed object applications by relieving the programmer or user of the burden of providing the large amounts of boilerplate computer code required for the basic housekeeping functions necessary for objects to be installed on the distributed object system. Components presented through the component catalog to the user can be manipulated graphically and placed into worksheets in which compositions comprising parts derived from the components contained in the catalog representing various existing distributed objects, modified distributed objects. or original distributed objects, are connected to define thereby new applications to be installed on the distributed object system. The use of the programmer information, and especially information relating to the plugs, sockets, and properties of the underlying objects represented by the components, is especially useful in such systems as the user then can graphically interconnect the plugs and sockets of various components (or their representations as parts) to define relationships among the distributed objects which will ultimately define the distributed object application without extensive examination of the underlying distributed object's code and the creation of large amounts of standard coding required to establish relationships among objects.

Furthermore, once the distributed application has been defined using the information available in the components and through their graphical interconnections *(e. g.,* as their derivative parts), a code generator facility (such as shown at 408 is Figure 4) then retrieves the references contained in the programmer information such as the factory IDL interface definitions, the names of methods used to acquire instances of objects *(i.e.* factory method names), service names for factories that are installed, header files, source code locations, macro definitions, and compiler information. to generate automatically the boilerplate code that is required to define the application as a distributed application and install that distributed application on the distributed object system.

Thus it will be seen from the foregoing that the present invention substantially addresses the needs of distributed object programmers for systems that facilitate the reuse and generation of code. Using the component services described herein, programmers can quickly identify and access distributed objects that have already been developed and installed on the distributed object system. The ability to easily browse and select existing distributed objects in a highly communal fashion as provided by the centralized library service and hierarchical directory structure described herein increases the sense of "community" among programmers by providing a central repository for their code which can be examined and incorporated into ongoing software development projects. By using components which not only provide information regarding the identity, composition, and use of the distributed objects which the components represent, but also include references required to construct the necessary boilerplate code to install software on a distributed object system, the tedious work of hand-coding such boilerplate is largely removed from the programmer; thus, allowing the programmer to focus on the more creative problems with respect to developing and installing software on a distributed object system.

It will be appreciated by those of skill in the programming arts, and especially the distributed programming arts, that the various methods and devices described herein can be implemented in a large variety of ways without departing from the scope of the present invention. In particular, it is noted that various hierarchical directory structures other than that illustrated in Figure 5 can be employed without departing from the present invention. In addition it is further noted that a variety of graphical user interfaces can be provided in addition to that shown in Figure 6 herein. For example, browser 604 can be adapted to show both the context hierarchy and the individual components. Also, additional windows

can be added to the catalog interface to provide view for both data sheets and component relationships in a substantially non-modal fashion. Finally the composition of components may include more, or less information than described herein.

Claims

1. A computer-implemented method for selecting and reviewing a distributed object installed on a distributed object system, comprising the steps of:

   a) generating a library of components corresponding to distributed objects on said distributed object system, said library of components including one component corresponding to said distributed object, wherein each of said components includes information describing the distributed object to which said component corresponds;

   b) displaying the contents of said library using a catalog interface device;

   c) browsing said library of components using said catalog interface device to identify said component corresponding to said distributed object;

   d) selecting said component corresponding to said object; and

   e) displaying at least a portion of said information describing said distributed object.

2. The computer-implemented method of claim 1, wherein said step of displaying the contents of said library comprises providing a display of at least a portion of the hierarchical directory structure of said library.

3. The computer-implemented method of claim 2, wherein said step of browsing said library of components comprises selecting a file path using said display of said hierarchical directory structure of said library and displaying representations of components located in said path.

4. The computer-implemented method of claim 3, wherein said representations comprise icons, and said step of selecting comprises making a selection action on an icon.

5. The computer-implemented method of any of claims 1-4, wherein said information includes presentation information, programmer information, and tool information for the object corresponding to said component.

6. The computer-implemented method of any of claims 1-5, further comprising the step of generating said component corresponding to said distributed object.

7. The computer-implemented method of claim 6, wherein said step of generating comprises activating a component service supported by said distributed object.

8. The computer-implemented method of claim 6, wherein said step of generating said component comprises generating said component when said object is installed on said distributed object system.

9. The computer-implemented method of claim 6, wherein said component is generated when said object is built.

10. A computer system for selecting and reviewing a distributed object installed on a distributed object system, comprising:

    a) a library of components corresponding to distributed objects on said distributed object system, said library of components including one component corresponding to said distributed object, wherein each of said components includes information describing the distributed object to which the component corresponds;

    b) a library manager which is effective to locate and retrieve said components: and

    c) a catalog interface device for displaying, reviewing and selecting said components in said library, said catalog interface device being coupled with said library manager such that queries and selection actions made in said catalog interface device are communicated to said library manager.

11. The computer system of claim 10, further comprising a component generator which is effective to generate components corresponding to selected, installed distributed objects in said distributed objects system.

12. The computer system of claim 11, wherein said component generator is coupled with the distributed objects to which said components refer.

13. The computer system of any of claims 11-12, further comprising an object development facility coupled with said component generator such that said components are generated when the distributed objects

to which the components refer are processed by said object development facility.

14. The computer system of any of claims 11-12, further comprising a builder coupled with said component generator such that said components are generated when the distributed objects to which the components refer are processed by said builder.

15. The computer system of any of claims 10-14, wherein said catalog interface device comprises a graphical user interface.

16. The computer system of claim 15, wherein said catalog interface devices comprises a display of at least a portion of the hierarchical directory structure of said library.

17. The computer system of claim 16, wherein said catalog interface device further includes a display of representations of said components located in a chose file path.

18. The computer system of claim 17, wherein said representations comprise icons.

19. The computer system of any of claims 10-18, wherein said components include presentation information, programmer information, and tool information for the object corresponding to said component.

20. A computer system comprising:

a) a processing unit;

b) an input/output device coupled with said processing unit;

c) random access memory coupled with said processing unit; and

d) a memory device in communication with said processing unit, said memory device including a distributed object reference data structure and an associated component data structure, said component data structure including presentation information, programmer information, and tool information relating to said distributed object data structure.

21. The computer system of claim 20, further comprising a library of components contained in a library data structure residing in said memory device, said library of components including a library service effective to locate and retrieve said components in response to commands received from said input/output device.

22. The computer system of claim 21, further comprising a component catalog coupled with said library service and said input/output device.

23. The computer system of claim 22, wherein said component catalog comprises a graphical user interface.

24. A computer program product comprising a computer-usable medium having computer-readable program code embodied thereon for a component data structure. said component data structure being related to a distributed object data structure installed on a distributed object system, and said component data structure including presentation information, programmer information, and tool information relating to said distributed object data structure.

25. The computer program product of claim 24, further comprising computer-usable medium having computer-readable program code embodied thereon for effecting the following steps within the computer system:

a) generating said component;

b) generating a library of component data structures corresponding to distributed object data structures on said distributed object system, wherein each of said component data structures includes information describing a distributed object data structure to which the component data structure corresponds;

c) displaying the contents of said library using a catalog interface device;

d) browsing said library of component data structures using said catalog interface device to identify said distributed object data structures; and

e) displaying at least a portion of said information describing said distributed object data structures.

26. The program product of claim 25, wherein said computer-readable program code generating includes code for activating a component service supported by said distributed object data structure.

27. The program product of claim 25, wherein computer-readable program code generating includes code for generating said component comprises generating said component data structure when said distributed object data structure is installed on said computer system.

28. The program product of claim 25, wherein said component data structure is generated when said distributed object data structure is built.

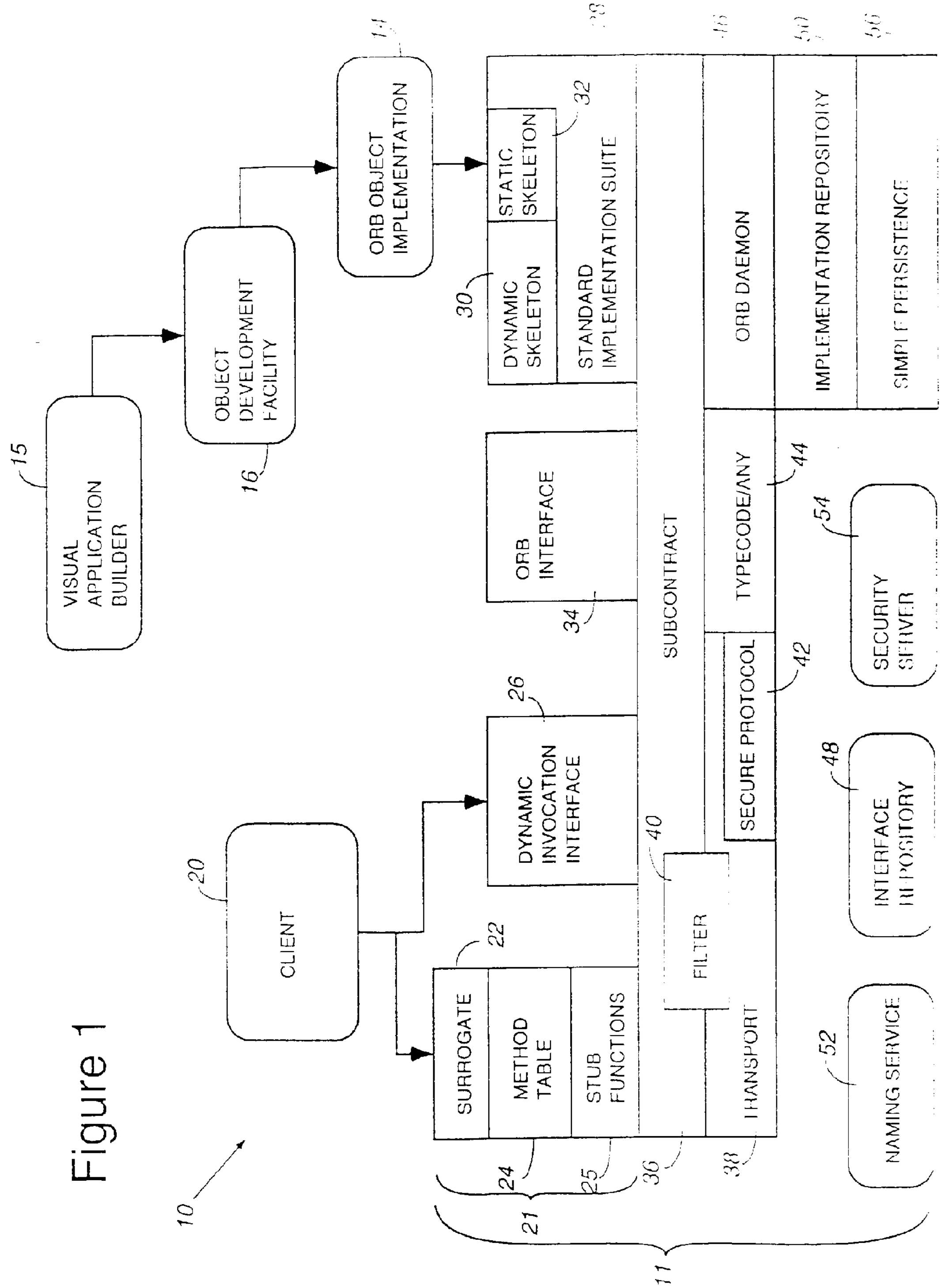
VISUAL APPLICATION BUILDER
15
OBJECT DEVELOPMENT FACILITY
16
ORB OBJECT IMPLEMENTATION
14
CLIENT
20
DYNAMIC INVOCATION INTERFACE
26
ORB INTERFACE
34
DYNAMIC SKELETON
30
STATIC SKELETON
32
STANDARD IMPLEMENTATION SUITE
28
SURROGATE
22
METHOD TABLE
24
STUB FUNCTIONS
25
21
SUBCONTRACT
36
FILTER
40
TRANSPORT
38
SECURE PROTOCOL
42
TYPECODE/ANY
44
ORB DAEMON
46
IMPLEMENTATION REPOSITORY
50
SIMPLE PERSISTENCE
56
SECURITY SERVER
54
INTERFACE REPOSITORY
48
NAMING SERVICE
52
10
11

Figure 1

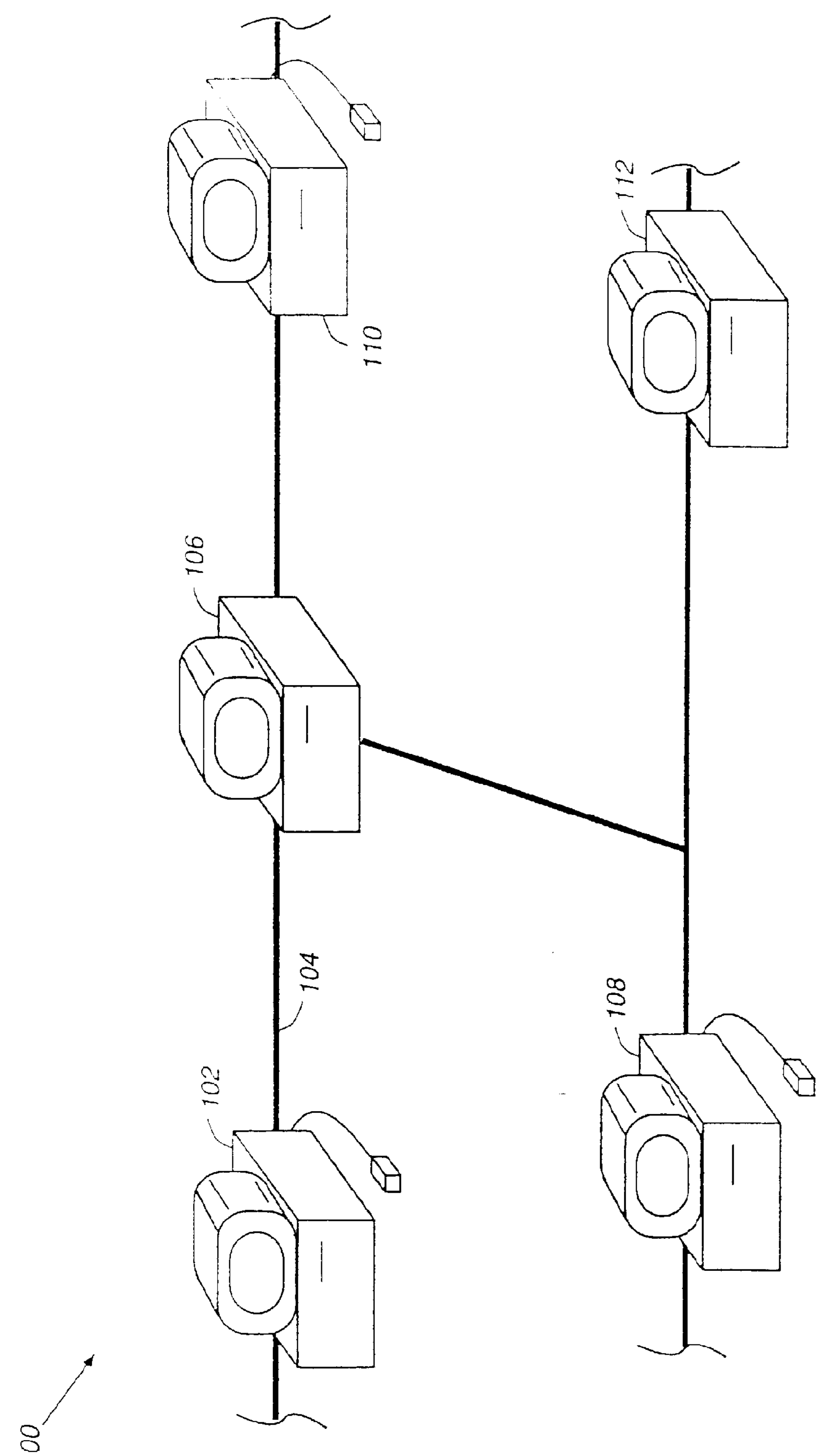

Figure 2

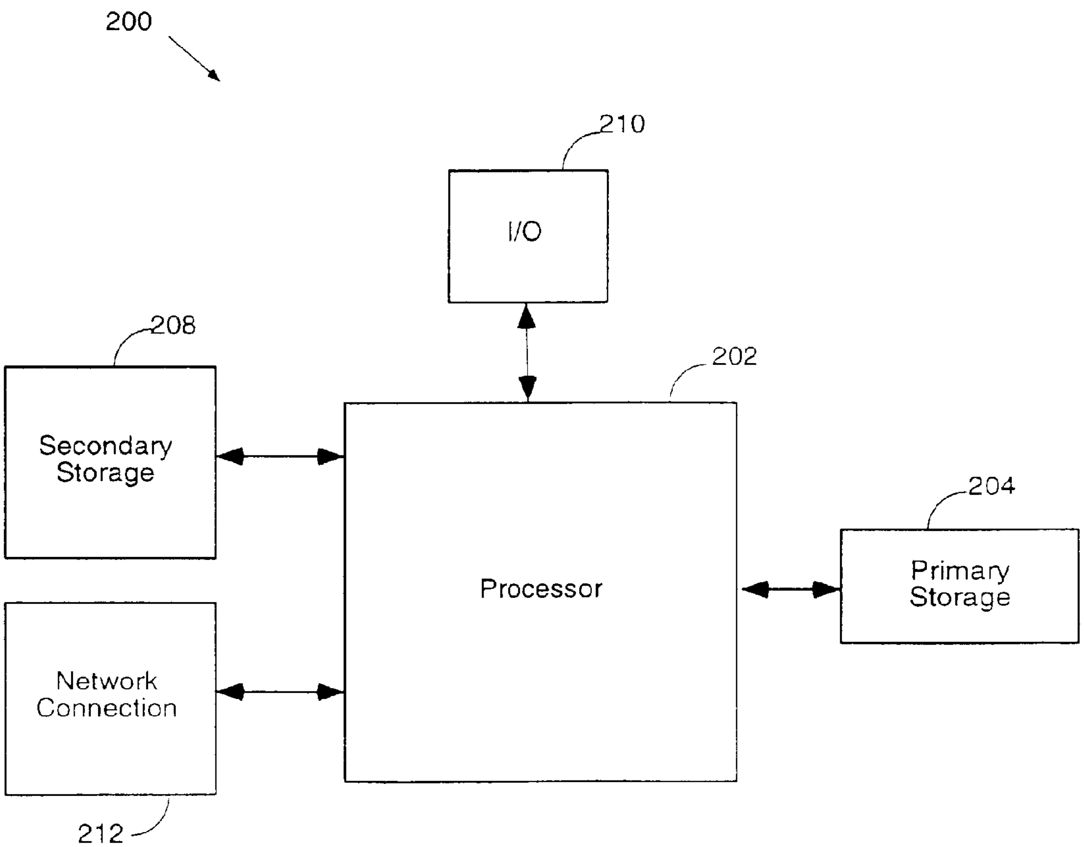
200
210
I/O
208
202
Secondary
Storage
204
Primary
Storage
Processor
Network
Connection
212

Figure 3

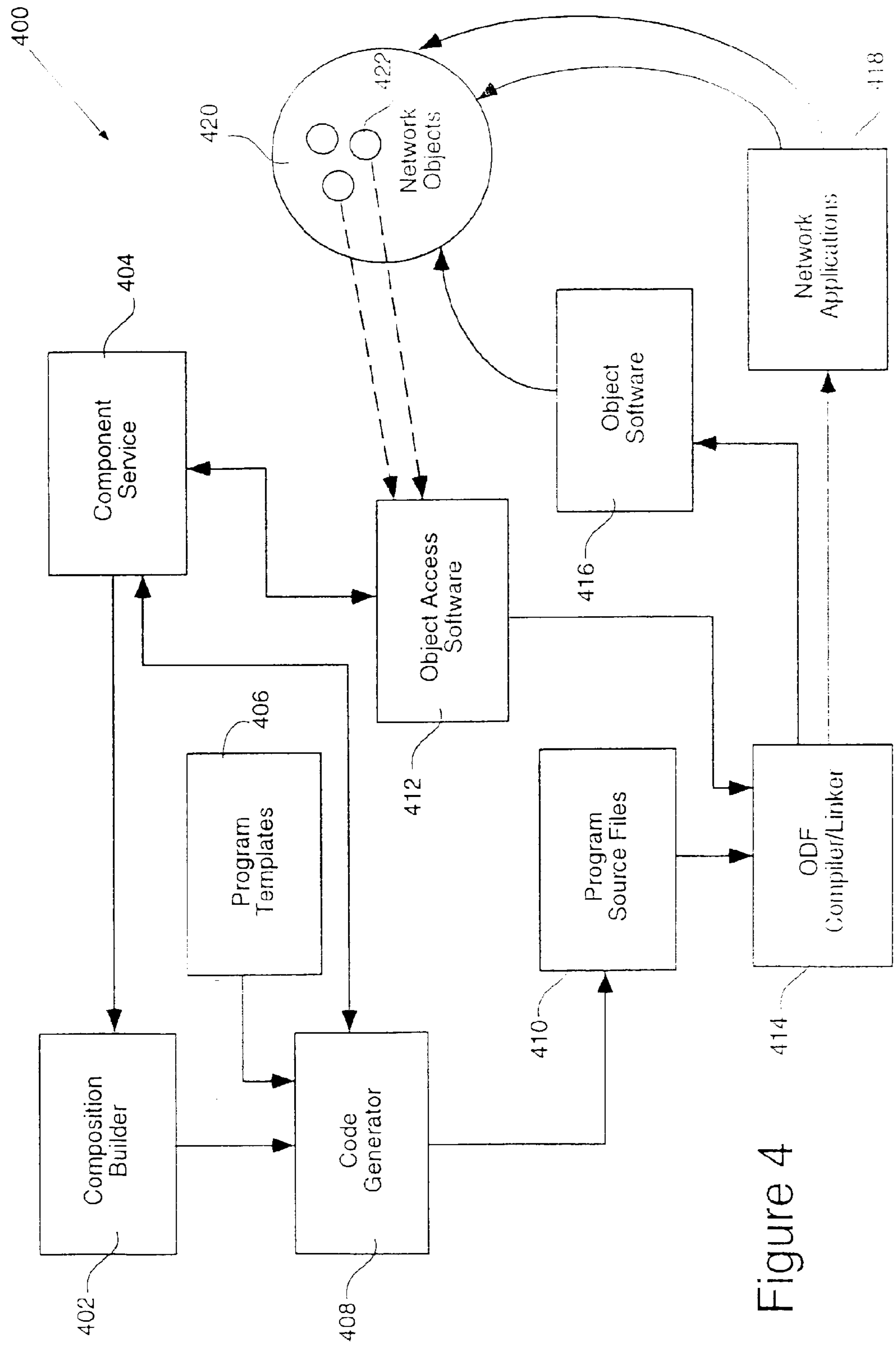
400
Composition Builder
402
Component Service
404
Program Templates
406
Code Generator
408
Program Source Files
410
Object Access Software
412
ODF Compiler/Linker
414
Object Software
416
Network Applications
418
Network Objects
420
422

Figure 4

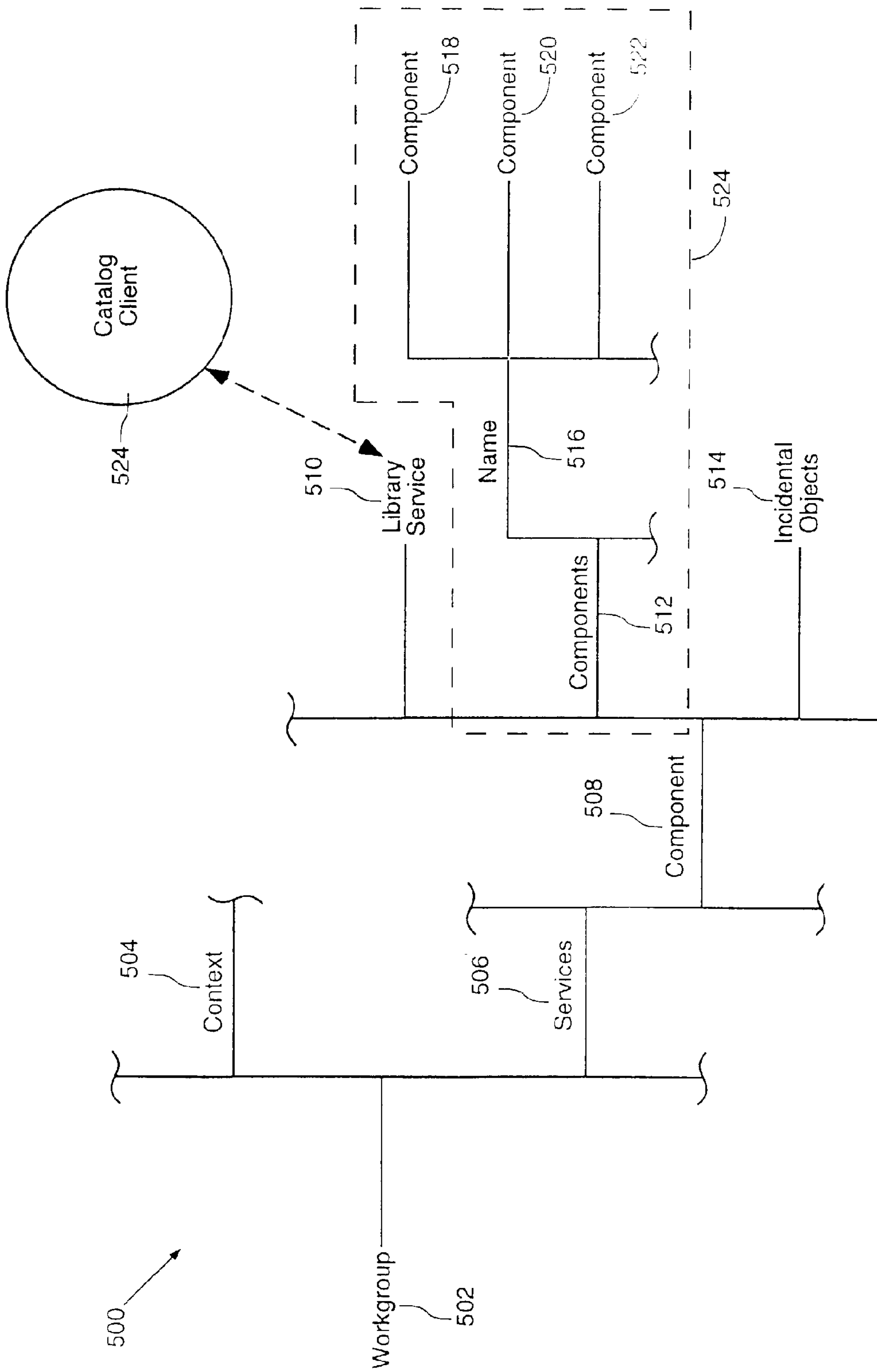
500
Workgroup
502
Context
504
Services
506
Component
508
Library Service
510
Components
512
Name
516
Incidental Objects
514
Component
518
Component
520
Component
522
524
Catalog Client
524

Figure 5

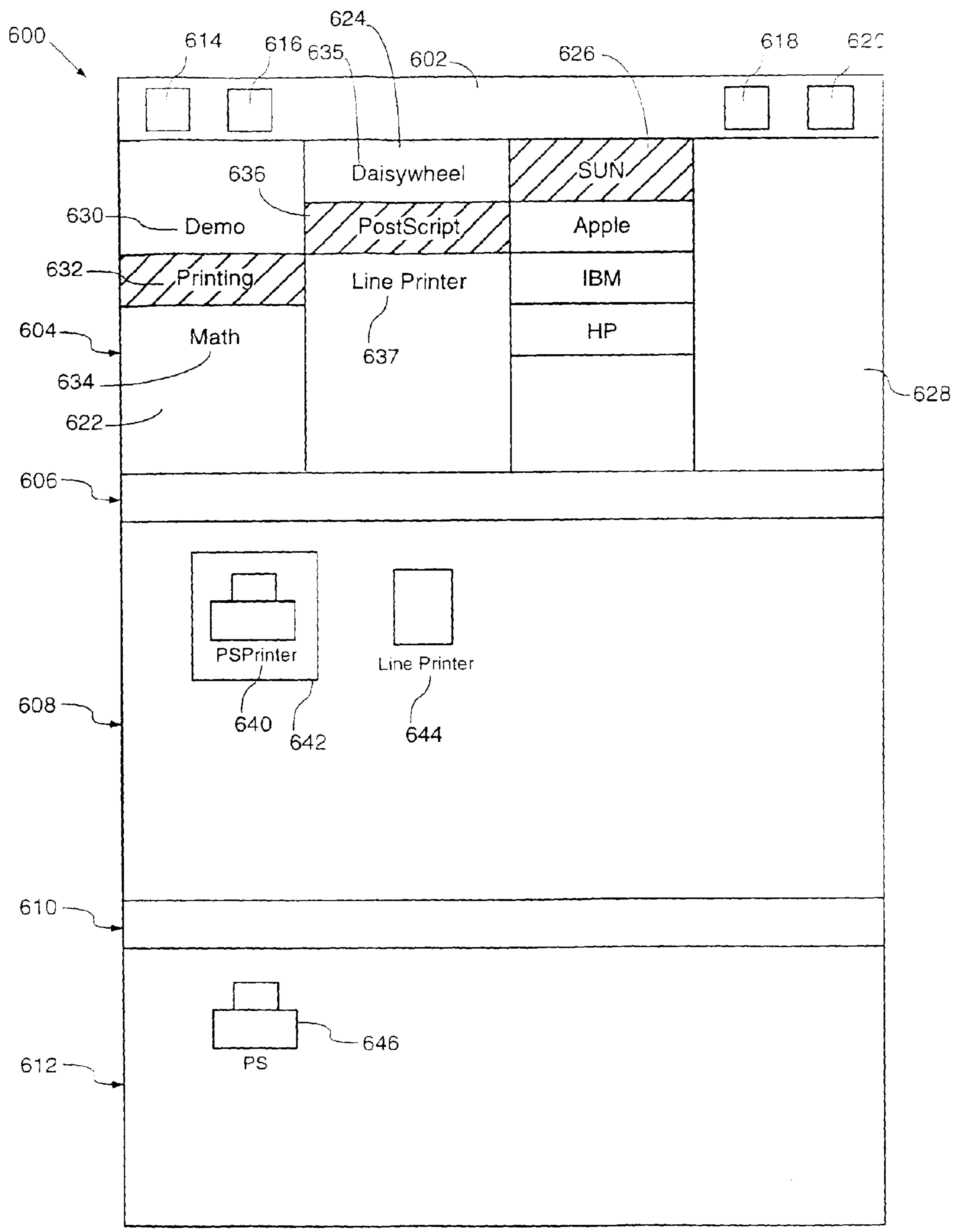
600
614
616
635
624
602
626
618
620
Daisywheel
SUN
636
630
Demo
PostScript
Apple
632
Printing
Line Printer
IBM
604
Math
HP
637
634
628
622
606
PSPrinter
Line Printer
608
640
642
644
610
646
PS
612

Figure 6